Journal of Machine Learning Research 10 (2009) 1239-1262 Submitted 10/01; Revised 10/08; Published 6/09

Incorporating Functional Knowledge in Neural Networks

Charles Dugas DUGAS@DMS.UMONTREAL.CA
Department of Mathematics and Statistic
Université de Montréal
2920 Chemin de la tour, suite 5190
Montreal, Qc, Canada H3T 1J4

Yoshua Bengio BENGIOY@IRO.UMONTREAL.CA
Department of Computer Science and Operations Research
Université de Montréal
2920 Chemin de la tour, suite 2194
Montreal, Qc, Canada H3A 1J4

François Bélisle BELISLE.FRANCOIS@GMAIL.COM
Claude Nadeau CLAUDE_NADEAU@HC-SC.GC.CA
Health Canada
Tunney's Pasture, PL 0913A
Ottawa, On, Canada K1A 0K9

René Garcia GARCIAR@CIRANO.QC.CA
CIRANO
2020 rue University, 25e étage
Montréal, Qc, Canada H3A 2A5

Editor: Peter Bartlett

Abstract

Incorporating prior knowledge of a particular task into the architecture of a learning algorithm can greatly improve generalization performance. We study here a case where we know that the function to be learned is non-decreasing in its two arguments and convex in one of them. For this purpose we propose a class of functions similar to multi-layer neural networks but (1) that has those properties, (2) is a universal approximator of Lipschitz[1] functions with these and other properties. We apply this new class of functions to the task of modelling the price of call options. Experiments show improvements on regressing the price of call options using the new types of function classes that incorporate the *a priori* constraints.

Keywords: neural networks, universal approximation, monotonicity, convexity, call options

1. Introduction

Incorporating *a priori* knowledge of a particular task into a learning algorithm helps reduce the necessary complexity of the learner and generally improves performance, if the incorporated knowledge is relevant to the task and brings enough information about the unknown generating process of the data. In this paper we consider prior knowledge on the positivity of some first and second derivatives of the function to be learned. In particular such constraints have applications to modelling

1. A function f is Lipschitz in Ω if $\exists c > 0$, $\forall x, y \in \Omega$, $|f(y) - f(x)| \leq c|y - x|$ (Delfour and Zolésio, 2001).

the price of stock options. Based on the Black-Scholes formula, the price of a call stock option is monotonically increasing in both the "moneyness" and time to maturity of the option, and it is convex in the "moneyness". Section 4 better explains these terms and stock options. For a function $f(x_1, x_2)$ of two real-valued arguments, this corresponds to the following properties:

$$f \geq 0, \qquad \frac{\partial f}{\partial x_1} \geq 0, \qquad \frac{\partial f}{\partial x_2} \geq 0, \qquad \frac{\partial^2 f}{\partial x_1^2} \geq 0. \tag{1}$$

The mathematical results of this paper (Section 2) are the following: first we introduce a class of one-argument functions that is positive, non-decreasing and convex in its argument. Second, we use this new class of functions as a building block to design another class of functions that is a universal approximator for functions with positive outputs. Third, once again using the first class of functions, we design a third class that is a universal approximator to functions of two or more arguments, with the set of arguments partitioned in two groups: those arguments for which the second derivative is known positive and those arguments for which we have no prior knowledge on the second derivative. The first derivative is positive for any argument. The universality property of the third class rests on additional constraints on cross-derivatives, which we illustrate below for the case of two arguments:

$$\frac{\partial^2 f}{\partial x_1 \partial x_2} \geq 0, \quad \frac{\partial^3 f}{\partial x_1^2 \partial x_2} \geq 0. \tag{2}$$

Thus, we assume that $f \in \mathcal{C}^3$, the set of functions three times continuously differentiable. Comparative experiments on these new classes of functions were performed on stock option prices, showing improvements when using these new classes rather than ordinary feedforward neural networks. The improvements appear to be non-stationary but the new class of functions shows the most stable behavior in predicting future prices. Detailed experimental results are presented in section 6.

2. Theory

Definition 1 *A class of functions $\hat{\mathcal{F}}$ from $\mathbb{R}^n$ to $\mathbb{R}$ is a* **universal approximator** *for a class of functions $\mathcal{F}$ from $\mathbb{R}^n$ to $\mathbb{R}$ if for any $f \in \mathcal{F}$, any compact domain $D \subset \mathbb{R}^n$, and any positive ε, one can find a $\hat{f} \in \hat{\mathcal{F}}$ with $\sup_{x \in D} |f(x) - \hat{f}(x)| \leq \varepsilon$.*

It has already been shown that the class of artificial neural networks with one hidden layer:

$$\hat{\mathcal{N}} = \left\{ f(x) = w_0 + \sum_{i=1}^{H} w_i \cdot h\left(b_i + \sum_j v_{ij} x_j\right) \right\}, \tag{3}$$

for example, with a sigmoid activation function $h(s) = 1/(1+e^{-s})$, is a universal approximator of continuous functions (Cybenko, 1988, 1989; Hornik et al., 1989; Barron, 1993). Furthermore, Leshno et al. (1993) have shown that any non-polynomial activation function will suffice for universal approximation. The number of hidden units H of the neural network is a hyper-parameter that controls the accuracy of the approximation and it should be chosen to balance the trade-off (see also Moody, 1994) between accuracy (bias of the class of functions) and variance (due to the finite sample used to estimate the parameters of the model). Because of this trade-off, in the finite sample case, it may be advantageous to consider a “simpler” class of functions that is appropriate to the task.

Since the sigmoid h is monotonically increasing ($h'(s) = h(s)(1-h(s)) > 0$), it is easy to force the first derivatives with respect to x to be positive by forcing the weights to be positive, for example with the exponential function:

$$\hat{\mathcal{N}}_+ = \left\{ f(x) = e^{w_0} + \sum_{i=1}^{H} e^{w_i} \cdot h\left(b_i + \sum_j e^{v_{ij}} x_j\right)\right\}. \tag{4}$$

Note that the positivity of $f(x)$ and $f'(x)$ is not affected by the values of the $\{b_i\}$ parameters. Since the sigmoid h has a positive first derivative, its primitive, which we call *softplus*, is convex:

$$\boxed{\zeta(s) = \ln(1+e^s)}$$

where $\ln(\cdot)$ is the natural logarithm operator. Note that $d\zeta(s)/ds = h(s) = 1/(1+e^{-s})$.

2.1 Universality for Functions with Positive Outputs

Using the softplus function introduced above, we define a new class of functions, all of which have positive outputs:

$$\hat{\mathcal{N}}_{>0} = \{f(x) = \zeta(g(x)), g(x) \in \hat{\mathcal{N}}\}.$$

Theorem 2 *Within the set of continuous functions from $\mathbb{R}^n$ to $\mathbb{R}_+ = \{x : x \in \mathbb{R}, x > 0\}$, the class $\hat{\mathcal{N}}_{>0}$ is a universal approximator.*

Proof Consider a positive function $f(x)$, which we want to approximate arbitrarily well. Consider $g(x) = \zeta^{-1}(f(x)) = \ln(e^{f(x)} - 1)$, the inverse softplus transform of $f(x)$. Choose $\hat{g}(x)$ from $\hat{\mathcal{N}}$ such that $\sup_{x\in D} |g(x) - \hat{g}(x)| \le \varepsilon$, where D is any compact domain over $\mathbb{R}^n$ and ε is any positive real number. The existence of $\hat{g}(x)$ is ensured by the universality property of $\hat{\mathcal{N}}$. Set $\hat{f}(x) = \zeta(\hat{g}(x)) = \ln(1+e^{\hat{g}(x)})$. Consider any particular x and define $a = \min(\hat{g}(x), g(x))$ and $b = \max(\hat{g}(x), g(x))$. Since $b - a \le \varepsilon$, we have,

$$\begin{aligned}
|\hat{f}(x) - f(x)| &= \ln(1+e^b) - \ln(1+e^a) \\
&= \ln\left(1 + (e^b - e^a)/(1+e^a)\right) \\
&\le \ln(1 + (e^\varepsilon - 1)e^a/(1+e^a)) \\
&< \varepsilon.
\end{aligned}$$

□

Thus, the use of the softplus function to transform the output of a regular one hidden layer artificial neural network ensures the positivity of the final output without hindering the universality property.

2.2 The Class ${}_{c,n}\hat{\mathcal{N}}_{++}$

In this section, we use the softplus function, in order to define a new class of functions with positive outputs, positive first derivatives w.r.t. all input variables and positive second derivatives w.r.t.

some of the input variables. The basic idea is to replace the sigmoid of a sum by a product of either softplus or sigmoid functions over each of the dimensions (using the softplus over the convex dimensions and the sigmoid over the others):

$$ {}_{c,n}\hat{\mathcal{N}}_{++} = \left\{ f(x) = e^{w_0} + \sum_{i=1}^{H} e^{w_i} \left(\prod_{j=1}^{c} \zeta(b_{ij} + e^{v_{ij}} x_j) \right) \left(\prod_{j=c+1}^{n} h(b_{ij} + e^{v_{ij}} x_j) \right) \right\}. \tag{5} $$

One can readily check that the output is necessarily positive, the first derivatives w.r.t. x_j are positive, and the second derivatives w.r.t. x_j for $j \le c$ are positive. However, this class of functions has other properties that are summarized by the following:

$$ \begin{aligned} \frac{\partial^m f}{\partial^{m_1} x_1 \partial^{m_2} x_2 \cdots \partial^{m_n} x_n} &\ge 0, \\ m_j &\in \begin{cases} \{0,1,2\} & 1 \le j \le c \\ \{0,1\} & c+1 \le j \le n, \end{cases} \\ \sum_{j=1}^{n} m_j &= m. \end{aligned} \tag{6} $$

Here, we have assumed that $f \in \mathcal{C}^{c+n}$, the set of functions that are $c+n$ times continuously differentiable. We will also restrict ourselves to Lispschitz functions since the proof of the theorem relies on the fact that the derivative of the function is bounded. The set of functions that respect these derivative conditions will be referred to as ${}_{c,n}\hat{\mathcal{F}}_{++}$. Note that, as special cases we find that f is positive ($m = 0$), and that it is monotonically increasing w.r.t. any of its inputs ($m = 1$), and convex w.r.t. the first c inputs ($m = 2, \exists j : m_j = 2$). Also note that, when applied to our particular case where $n = 2$ and $c = 1$, this set of equations corresponds to Equations (1) and (2). We now state the main universality theorem:

Theorem 3 *Within the set ${}_{c,n}\hat{\mathcal{F}}_{++}$ of Lipschitz functions from $\mathbb{R}^n$ to $\mathbb{R}$ whose set of derivatives as specified by Equation* (6) *are non-negative, the class ${}_{c,n}\hat{\mathcal{N}}_{++}$ is a universal approximator.*

The proof of the theorem is given in Section A.

2.3 Parameter Optimization

In our experiments, conjugate gradient descent was used to optimize the parameters of the model. The backpropagation equations are obtained as the derivatives of $f \in {}_{c,n}\hat{\mathcal{N}}_{++}$ (Equation 5) w.r.t. to its parameters. Let $z_{i,j} = b_{ij} + e^{v_{ij}} x_j$, $u_i = e^{w_i} (\prod_{j=1}^{c} \zeta(z_{i,j}))(\prod_{j=c+1}^{n} h(z_{ij}))$ and $f = e^{w_0} + \sum_{i=1}^{H} u_i$. Then, we have

$$ \begin{aligned} \partial f / \partial w_0 &= e^{w_0}, \\ \partial f / \partial w_i &= u_i, \\ \partial f / \partial b_{i,k} &= \begin{cases} u_i \cdot h(z_{i,k}) / \zeta(z_{i,k}) & 1 \le k \le c \\ u_i \cdot (1 - h(z_{i,k})) & c+1 \le k \le n, \end{cases} \\ \partial f / \partial v_{i,k} &= e^{v_{ik}} x_k \cdot \partial f / \partial b_{i,k}. \end{aligned} \tag{7} $$

Except for terms $h(z_{i,k}), k \le c$ of Equation (7), all values are computed through the forward phase, that is, while computing the value of f. Error backpropagation can thus be performed efficiently if careful attention is paid, during the forward phase, to store the values to be reused in the backpropagation phase.

Software implementing parameter optimization of the proposed architecture and the numerical experiments of the following section is available on-line.[2] Code was written using the "R" statistical software package.[3]

3. Experiments with Artificial Data

In this section, we present a series of controlled experiments in order to assess the potential improvements that can be gained from using the proposed architecture in cases where some derivatives of the target function are known to be positive. The emphasis is put on analyzing the evolution of the model bias and model variance values w.r.t. various noise levels and training set sizes.

The function we shall attempt to learn is

$$\begin{aligned} f(\vec{x}) &= \zeta(x_1)\zeta(x_2)\zeta(x_3)h(x_4), \\ y &= f(\vec{x}) + \xi, \end{aligned}$$

where $\zeta(\cdot)$ is the softplus function defined above and $h(\cdot)$ is the sigmoid function. The input values are drawn from a uniform distribution over the [0,1] interval, that is, $x_i \sim \mathcal{U}(0,1)$. The noise term ξ is added to the true function $f(\vec{x})$ to generate the target value y. Finally, $\xi \sim \mathcal{N}(0,\sigma^2)$, that is, we used additive Gaussian noise. Different values for σ have been tested.

For each combination of noise level ($\sigma \in \{$1e-2, 3e-2, 1e-1$\}$) and training set size (25, 50, 100, 200, 400), we chose the best performing combination of number of hidden units and weight decay. In order to perform model selection, 100 models were trained using different random training sets, for each combination. Based on validation set performance, 50 models were retained and their validation set performances were averaged. The best performing combination was chosen based on this average validation performance. Bias and variance were measured using these 50 selected models when applied on another testset of 10000 examples. In each case, the number of training epochs was 10000. The process was repeated for two architectures: the proposed architecture of products of softplus and sigmoid functions over input dimensions with constrained weights (CPSD) and regular unconstrained multi-layered perceptrons with a single hidden layer (UMLP).

In order to compute the bias and variance values, we first computed, for each test example, the average of the $N_D = 50$ model outputs:

$$\bar{g}(\vec{x}_i) = \frac{1}{N_D}\sum_{j=1}^{N_D} g_j(\vec{x}_i),$$

where $g_j(\vec{x}_i)$ is the output of the j^{th} model associated to the i^{th} input vector $\vec{x}_i$.

The variance was unbiasedly approximated as the average over all test examples ($N_i = 10000$), of the sample variance of model outputs $g_j(\vec{x}_i)$ w.r.t. the corresponding mean output $\bar{g}(\vec{x}_i)$:

$$\begin{aligned} \hat{v}(\vec{x}_i) &= \frac{1}{N_D - 1}\sum_{j=1}^{N_D}(g_j(\vec{x}_i) - \bar{g}(\vec{x}_i))^2, \\ \hat{v} &= \frac{1}{N_i}\sum_i \hat{v}(\vec{x}_i). \end{aligned}$$

2. Software can be found at `http://www.dms.umontreal.ca/~dugas/convex/`.
3. Code found at `http://www.r-project.org/`.

The bias was unbiasedly estimated as the average over all test examples, of the squared deviation of the mean output $\bar{g}(\vec{x}_i)$ w.r.t. the known true function value $f(\vec{x}_i)$, less a variance term:

$$\begin{aligned}\hat{b}(\vec{x}_i) &= (\bar{g}(\vec{x}_i) - f(\vec{x}_i))^2 - \hat{v}(\vec{x}_i)/N_D,\\ \hat{b} &= \frac{1}{N_i}\sum_i \hat{b}(\vec{x}_i).\end{aligned}$$

Let $b(\vec{x}_i) = (E_G(g(\vec{x}_i)) - f(\vec{x}_i))^2$ be the true bias, at point $\vec{x}_i$ where $E_G()$ denotes expectation taken over training set distribution, which induces a distribution of the function g produced by the learning algorithm. Let us show that $E_G(\hat{b}(\vec{x}_i)) = b(\vec{x}_i)$:

$$\begin{aligned}E_G(\hat{b}(\vec{x}_i)) &= E_G[(\bar{g}(\vec{x}_i) - f(\vec{x}_i))^2 - \hat{v}(\vec{x}_i)/N_D],\\ &= E_G[(\bar{g}(\vec{x}_i) - g(\vec{x}_i) + g(\vec{x}_i) - f(\vec{x}_i))^2] - v(\vec{x}_i)/N_D,\\ &= E_G[(\bar{g}(\vec{x}_i) - g(\vec{x}_i))^2] + E_G[(g(\vec{x}_i) - f(\vec{x}_i))^2] - v(\vec{x}_i)/N_D,\\ &= v(\vec{x}_i)/N_D + b(\vec{x}_i) - v(\vec{x}_i)/N_D,\\ &= b(\vec{x}_i).\end{aligned}$$

Table 1 reports the results for these simulations. In all cases, the bias and variance are lower for the proposed architecture than for a regular neural network architecture, which is the result we expected. The variance reduction is easy to understand because of the appropriate constraints on the class of functions. The bias reduction, we conjecture to be a side effect of the bias-variance tradeoff being performed by the model selection on the validation set: to achieve a lower validation error, a larger bias is needed with the unconstrained artificial neural network. The improvements are generally more important for smaller sample sizes. A possible explanation is that the proposed architecture helps reduce the variance of the estimator. With small sample sizes, this is very beneficial and becomes less important as the number of points increases.

4. Estimating Call Option Prices

An option is a contract between two parties that entitles the buyer to a claim at a future date T that depends on the future price, S_T of an underlying asset whose price at current time t is S_t. In this paper we consider the very common European call options, in which the buyer (holder) of the option obtains the right to buy the asset at a fixed price K called the strike price. This purchase can only occur at maturity date (time T). Thus, if at maturity, the price of the asset S_T is above the strike price K, the holder of the option can *exercise* his option and buy the asset at price K, then sell it back on the market at price S_T, thus making a profit of $S_T - K$. If, on the other hand, the price of the asset at maturity S_T is below the strike price K, then the holder of the option has no interest in exercising his option (and does not have to) and the option simply expires worthless and unexercised. For this reason, the option is considered to be worth $\max(0, S_T - K)$ at maturity and our goal is to estimate C_t, the value of that worth at current time t.

In the econometric literature, the call function is often expressed in terms of the primary economic variables that influence its value: the actual market price of the security (S_t), the strike price (K), the remaining time to maturity ($\tau = T - t$), the risk free interest rate (r), and the volatility of the return (σ). One important result is that under mild conditions, the call option function is homogeneous of degree one with respect to the strike price and so we can perform dimensionality reduction

Bias and Variance Analysis on Artificial Data

Ntrain	Noise	Architecture	Bias	Variance	Sum
25	1e-02	UMLP	2.31e-04	9.20e-05	3.23e-04
		CPSD	**1.04e-04**	**3.97e-05**	**1.43e-04**
	3e-02	UMLP	1.06e-03	3.46e-04	1.40e-03
		CPSD	**9.37e-04**	**2.30e-04**	**1.17e-03**
	1e-01	UMLP	1.07e-02	2.68e-03	1.33e-02
		CPSD	**1.02e-02**	**2.45e-03**	**1.27e-02**
50	1e-02	UMLP	1.55e-04	9.41e-05	2.49e-04
		CPSD	**1.03e-04**	**1.99e-05**	**1.23e-04**
	3e-02	UMLP	1.05e-03	1.28e-04	1.18e-03
		CPSD	**9.35e-04**	**9.29e-05**	**1.03e-03**
	1e-01	UMLP	1.03e-02	1.22e-03	1.15e-02
		CPSD	**1.02e-02**	**1.11e-03**	**1.13e-02**
100	1e-02	UMLP	1.27e-04	3.98e-05	1.67e-04
		CPSD	**1.02e-04**	**1.01e-05**	**1.12e-04**
	3e-02	UMLP	9.82e-04	2.11e-04	1.19e-03
		CPSD	**9.39e-04**	**4.77e-05**	**9.87e-04**
	1e-01	UMLP	1.04e-02	6.28e-04	1.10e-02
		CPSD	**1.02e-02**	**5.30e-04**	**1.07e-02**
200	1e-02	UMLP	1.07e-04	2.24e-05	1.29e-04
		CPSD	**1.02e-04**	**5.01e-06**	**1.07e-04**
	3e-02	UMLP	9.45e-04	1.10e-04	1.05e-03
		CPSD	**9.15e-04**	**4.31e-05**	**9.58e-04**
	1e-01	UMLP	1.03e-02	3.38e-04	1.07e-02
		CPSD	**1.02e-02**	**3.21e-04**	**1.05e-02**
400	1e-02	UMLP	1.03e-04	1.14e-05	1.15e-04
		CPSD	**1.02e-04**	**2.41e-06**	**1.04e-04**
	3e-02	UMLP	9.32e-04	6.15e-05	9.94e-04
		CPSD	**9.15e-04**	**2.10e-05**	**9.36e-04**
	1e-01	UMLP	1.04e-02	1.75e-04	1.05e-02
		CPSD	**1.02e-02**	**1.43e-04**	**1.03e-02**

Table 1: Comparison of the bias and variance values for two neural network architectures, three levels of noise, and five sizes of training sets (Ntrain), using artificial data. In bold, the best performance between the two models.

by letting our approximating function depend on the "moneyness" ratio ($M = S_t/K$) instead of the current asset price S_t and the strike price K independently. We must then modify the target to be the price of the option divided by the strike price: C_t/K.

Most of the research on call option modelling relies on strong parametric assumptions of the underlying asset price dynamics. Any misspecification of the stochastic process for the asset price will lead to systematic mispricings for any option based on the asset (Hutchinson et al., 1994). The well-known Black-Scholes formula (Black and Scholes, 1973) is a consequence of such specifications and other assumptions:

$$f(M,\tau,r,\sigma) = M\Phi(d_1) - e^{-r\tau}\Phi(d_2),$$

where $\Phi(\cdot)$ is the cumulative Gaussian function evaluated in points

$$d_1, d_2 = \frac{\ln M + (r \pm \sigma^2/2)\tau}{\sigma\sqrt{\tau}},$$

that is, $d_1 = d_2 + \sigma\sqrt{\tau}$. In particular, two assumptions on which this formula relies have been challenged by empirical evidence: the assumed lognormality of returns on the asset and the assumed constance of volatility over time.

On the other hand, nonparametric models such as neural networks do not rely on such strong assumptions and are therefore robust to model specification errors and their consequences on option modelling and this motivates research in the direction of applying nonparametric techniques for option modelling.

Analyzing the primary economic variables that influence the call option price, we note that the risk free interest rate (r) needs to be somehow extracted from the term structure of interest rates and the volatility (σ) needs to be forecasted. This latter task is a field of research in itself. Dugas et al. (2000) have previously tried to feed in neural networks with estimates of the volatility using historical averages but the gains have remained insignificant. We therefore drop these two features and rely on the ones that can be observed (S_t, K, τ) to obtain the following:

$$C_t/K = f(M,\tau).$$

The novelty of our approach is to account for properties of the call option function as stated in Equation (1). These properties derive from simple arbitrage pricing theory.[4] Now even though we know the call option function to respect these properties, we do not know if it does respect the additional cross derivative properties of Equation (2). In order to gain some insight in this direction, we confront the Black-Scholes formula to our set of constraints:

$$\frac{\partial f}{\partial M} = \Phi(d_1), \tag{8}$$

$$\frac{\partial^2 f}{\partial M^2} = \frac{\phi(d_1)}{\sqrt{\tau}M\sigma}, \tag{9}$$

$$\frac{\partial f}{\partial \tau} = e^{-r\tau}\left(\frac{\phi(d_2)\sigma}{2\sqrt{\tau}} + r\Phi(d_2)\right), \tag{10}$$

$$\frac{\partial^2 f}{\partial M \partial \tau} = \frac{\phi(d_1)}{2\sigma\tau^{3/2}}\left((r+\sigma^2/2)\tau - \ln M\right), \tag{11}$$

$$\frac{\partial^3 f}{\partial M^2 \partial \tau} = \frac{\phi(d_1)}{2M\sigma^3\tau^{5/2}}\left(\ln^2 M - \sigma^2\tau - (r+\sigma^2/2)^2\tau^2\right), \tag{12}$$

where $\phi(\cdot)$ is the Gaussian density function. Equations (8), (9) and (10) confirm that the Black-Scholes formula is in accordance with our prior knowledge of the call option function: all three

4. The convexity of the call option w.r.t. the moneyness is a consequence of the butterfly spread strategy (Garcia and Gençay, 1998).

derivatives are positive. Equations (11) and (12) are the cross derivatives which will be positive for any function chosen from ${}_{1,2}\hat{\mathcal{N}}_{++}$. When applied to the Black-Scholes formula, it is less clear whether these values are positive, too. In particular, one can easily see that both cross derivatives can not be simultaneously positive. Thus, the Black-Scholes formula is not within the set ${}_{1,2}\hat{\mathcal{F}}_{++}$. Then again, it is known that the Black-Scholes formula does not adequately represent the market pricing of options, but it is considered as a useful guide for evaluating call option prices. So, we do not know if these constraints on the cross derivatives are present in the true price function.

Nonetheless, even if these additional constraints are not respected by the true function on all of its domain, one can hope that the increase in the bias of the estimator due to the constraints will be offset (because we are searching in a smaller function space) by a decrease in the variance of that estimator and that overall, the mean-squared error will decrease. This strategy has often been used successfully in machine learning (e.g., regularization, feature selection, smoothing).

5. Experimental Setup

As a reference model, we use a simple multi-layered perceptron with one hidden layer (Equation 3). For **UMLP models**, weights are left unconstrained whereas for **CMLP models**, weights are constrained, through exponentiation, to be positive. We also compare our results with a recently proposed model (Garcia and Gençay, 1998) that closely resembles the Black-Scholes formula for option pricing (i.e., another way to incorporate possibly useful prior knowledge):

$$\begin{aligned} y &= \alpha + M \cdot \sum_{i=1}^{n_h} \beta_{1,i} \cdot h(\gamma_{i,0} + \gamma_{i,1} \cdot M + \gamma_{i,2} \cdot \tau) \\ &+ e^{-r\tau} \cdot \sum_{i=1}^{n_h} \beta_{2,i} \cdot h(\gamma_{i,3} + \gamma_{i,4} \cdot M + \gamma_{i,5} \cdot \tau), \end{aligned} \tag{13}$$

with inputs M, τ, parameters r, α, β, γ and hyperparameter n_h (number of hidden units). We shall refer to Equation (13) as the **UBS models**. Constraining the weights of Equation (13) through exponentiation leads to a different architecture we refer to as the **CBS models**.

We evaluate two new architectures incorporating all of the constraints defined in Equation (6). The proposed architecture involves the product of softplus and sigmoid functions over input dimensions, hence the **UPSD models** and **CPSD models** labels for an unconstrained version of the proposed architecture and the proposed constrained architecture, respectively. Finally, we also tested another architecture derived from the proposed one by simply summing, instead of multiplying, softplus and sigmoid functions. For that last architecture (with constrained weights), positivity, monotonicity and convexity properties are respected but in that case, cross-derivatives are all equal to zero. We do not have a universality proof for that specific class of functions. The unconstrained and constrained architectures are labelled as **USSD models** and **CSSD models**, respectively.

We used European call option data from 1988 to 1993. A total of 43518 transaction prices on European call options on the S&P500 index were used. In Section 6, we report results on 1988 data. In each case, we used the first two quarters of 1988 as a training set (3434 examples), the third quarter as a validation set (1642 examples) for model selection and the fourth quarter as a test set (each with around 1500 examples) for final generalization error estimation. In tables 2 and 3, we present results for networks with unconstrained weights on the left-hand side, and weights constrained to positive and monotone functions through exponentiation of parameters on the right-hand side. For each model, the number of hidden units varies from one to nine. The mean squared error results reported were obtained as follows: first, we randomly sampled the parameter space 1000 times. We picked the best (lowest training error) model and trained it up to 1000 more epochs. Repeating this procedure 10 times, we selected and averaged the performance of the best of these 10 models (those with training error no more than 10% worse than the best out of 10). In figure 1, we present tests of the same models on each quarter up to and including 1993 (20 additional test sets) in order to assess the persistence (conversely, the degradation through time) of the trained models.

6. Forecasting Results

As can be seen in tables 2 and 3, unconstrained architectures obtain better training, validation and testing (test 1) results but fail in the extra testing set (test 2). A possible explanation is that constrained architectures capture more fundamental relationships between variables and are more robust to nonstationarities of the underlying process. Constrained architectures therefore seem to give better generalization when considering longer time spans.

The importance in the difference in performance between constrained and unconstrained architectures on the second test set lead us to look even farther into the future and test the selected models on data from later years. In Figure 1, we see that the Black-Scholes similar constrained model performs slightly better than other models on the second test set but then fails on later quarters. All in all, at the expense of slightly higher initial errors our proposed architecture allows one to forecast with increased stability much farther in the future. This is a very welcome property as new derivative products have a tendency to lock in values for much longer durations (up to 10 years) than traditional ones.

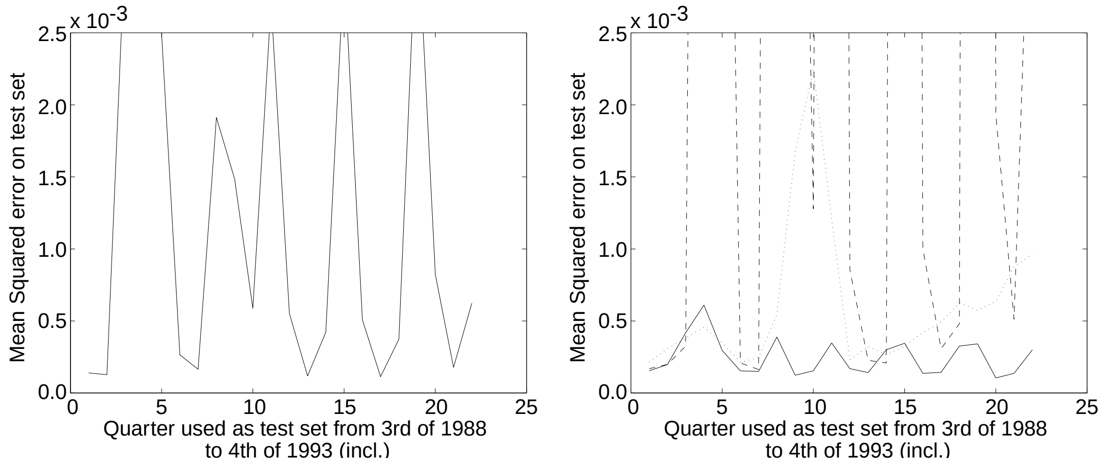

Figure 1: Out-of-sample results from the third quarter of 1988 to the fourth of 1993 (incl.) for models with best validation results. Left: unconstrained models; results for the UBS models. Other unconstrained models exhibit similar swinging result patterns and levels of errors. Right: constrained models. The proposed CPSD architecture (solid) does best. The model with sums over dimensions (CSSD) obtains similar results. Both CMLP (dotted) and CBS (dashed) models obtain poorer results. (dashed).

Mean Squared Error Results on Call Option Pricing ($\times 10^{-4}$)

Hidden Units	UMLP				CMLP			
	Train	Valid	Test1	Test2	Train	Valid	Test1	Test2
1	2.38	1.92	2.73	6.06	2.67	2.32	3.02	3.60
2	1.68	1.76	1.51	5.70	2.63	2.14	3.08	3.81
3	1.40	1.39	1.27	27.31	2.63	2.15	3.07	3.79
4	1.42	1.44	1.25	27.32	2.65	2.24	3.05	3.70
5	1.40	1.38	**1.27**	**30.56**	2.67	2.29	3.03	3.64
6	1.41	1.43	1.24	33.12	2.63	2.14	**3.08**	**3.81**
7	1.41	1.41	1.26	33.49	2.65	2.23	3.05	3.71
8	1.41	1.43	1.24	39.72	2.63	2.14	3.07	3.80
9	1.40	1.41	1.24	38.07	2.66	2.27	3.04	3.67

Hidden Units	UBS				CBS			
	Train	Valid	Test1	Test2	Train	Valid	Test1	Test2
1	1.54	1.58	1.40	4.70	2.49	2.17	2.78	3.61
2	1.42	1.42	1.27	24.53	1.90	1.71	2.05	3.19
3	1.40	1.41	1.24	30.83	1.88	1.73	2.00	3.72
4	1.40	1.39	**1.27**	**31.43**	1.85	1.70	1.96	3.15
5	1.40	1.40	1.25	30.82	1.87	1.70	2.01	3.51
6	1.41	1.42	1.25	35.77	1.89	1.70	2.04	3.19
7	1.40	1.40	1.25	35.97	1.87	1.72	1.98	3.12
8	1.40	1.40	1.25	34.68	1.86	1.69	**1.98**	**3.25**
9	1.42	1.43	1.26	32.65	1.92	1.73	2.08	3.17

Table 2: Left: the parameters are free to take on negative values. Right: parameters are constrained through exponentiation so that the resulting function is both positive and monotone increasing everywhere w.r.t. both inputs as in Equation (4). Top: regular feedforward artificial neural networks. Bottom: neural networks with an architecture resembling the Black-Scholes formula as defined in Equation (13). The number of hidden units varies from 1 to 9 for each network architecture. The first two quarters of 1988 were used for training, the third of 1988 for validation and the fourth of 1988 for testing. The first quarter of 1989 was used as a second test set to assess the persistence of the models through time (figure 1). In bold: test results for models with best validation results.

In another series of experiments, we tested the unconstrained multi-layered perceptron against the proposed constrained products of softplus convex architecture using data from years 1988 through 1993 incl. For each year, the first two quarters were used for training, the third quarter for model selection (validation) and the fourth quarter for testing. We trained neural networks for 50000 epochs and with a number of hidden units ranging from 1 through 10. In Table 4, we report training, validation and test results for the two chosen architectures. Model selection was performed using the validation set in order to choose the best number of hidden units, learning rate, learning rate decrease and weight decay. In all cases, except for 1988, the proposed architecture outperformed the multi-layered perceptron model. This might explain why the proposed architecture did

Mean Squared Error Results on Call Option Pricing ($\times 10^{-4}$)

Hidden Units	UPSD				CPSD			
	Train	Valid	Test1	Test2	Train	Valid	Test1	Test2
1	2.27	2.15	2.35	3.27	2.28	2.14	2.37	3.51
2	1.61	1.58	1.58	14.24	2.28	2.13	2.37	3.48
3	1.51	1.53	1.38	18.16	2.28	2.13	2.36	3.48
4	1.46	1.51	1.29	20.14	1.84	1.54	**1.97**	**4.19**
5	1.57	1.57	1.46	10.03	1.83	1.56	1.95	4.18
6	1.51	1.53	1.35	22.47	1.85	1.57	1.97	4.09
7	1.62	1.67	1.46	7.78	1.86	1.55	2.00	4.10
8	1.55	1.54	1.44	11.58	1.84	1.55	1.96	4.25
9	1.46	1.47	**1.31**	**26.13**	1.87	1.60	1.97	4.12

Hidden Units	USSD				CSSD			
	Train	Valid	Test1	Test2	Train	Valid	Test1	Test2
1	1.83	1.59	1.93	4.10	2.30	2.19	2.36	3.43
2	1.42	1.45	**1.26**	**25.00**	2.29	2.19	2.34	3.39
3	1.45	1.46	1.32	35.00	1.84	1.58	1.95	4.11
4	1.56	1.69	1.33	21.80	1.85	1.56	1.99	4.09
5	1.60	1.69	1.42	10.11	1.85	1.52	**2.00**	**4.21**
6	1.57	1.66	1.39	14.99	1.86	1.54	2.00	4.12
7	1.61	1.67	1.48	8.00	1.86	1.60	1.98	3.94
8	1.64	1.72	1.48	7.89	1.85	1.54	1.98	4.25
9	1.65	1.70	1.52	6.16	1.84	1.54	1.97	4.25

Table 3: Similar results as in table 2 but for two new architectures. Top: products of softplus along the convex axis with sigmoid along the monotone axis. Bottom: the softplus and sigmoid functions are summed instead of being multiplied. Top right: the fully constrained proposed architecture (CPSD).

not perform as well as other architectures on previous experiments using only data from 1988. Also note that the MSE obtained in 1989 is much higher. This is a possible explanation for the bad results obtained in tables 2 and 3 on the second test set. A hypothesis is that the process was undergoing nonstationarities that affected the forecasting performances. This shows that performance can vary by an order of magnitude from year to year and that forecasting in the presence of nonstationary processes is a difficult task.

7. Conclusions

Motivated by prior knowledge on the positivity of the derivatives of the function that gives the price of European options, we have introduced new classes of functions similar to multi-layer neural networks that have those properties. We have shown universal approximation properties for these classes. On simulation experiments, using artificial data sets, we have shown that these classes of functions lead to a reduction in the variance and the bias of the associated estimators. When applied

Mean Squared Error Results
on Call Option Pricing ($\times 10^{-5}$)

Year	Architecture	Units	Train	Valid	Test
1988	UMLP	9	2.09	1.45	3.28
	CPSD	9	3.86	2.70	5.23
1989	UMLP	4	9.10	28.89	51.39
	CPSD	2	9.31	23.96	48.22
1990	UMLP	9	2.17	4.81	5.61
	CPSD	9	1.58	4.18	5.39
1991	UMLP	9	2.69	1.76	3.41
	CPSD	8	2.62	1.25	2.74
1992	UMLP	8	3.46	1.16	1.52
	CPSD	8	3.27	1.28	1.27
1993	UMLP	9	1.34	1.47	1.76
	CPSD	10	0.68	0.54	0.65

Table 4: Comparison between a simple unconstrained multi-layered architecture (UMLP) and the proposed architecture (CPSD). Data from the first two quarters of each year was used as training set, data from the third quarter was used for validation and the fourth quarter was used for testing. We also report the number of units chosen by the model selection process.

in empirical tests of option pricing, we showed that the architecture from the proposed constrained classes usually generalizes better than a standard artificial neural network.

Appendix A. Proof of the Universality Theorem for Class ${}_{c,n}\hat{\mathcal{N}}_{++}$

In this section, we prove theorem 2.2. In order to help the reader through the formal mathematics, we first give an outline of the proof, that is, a high-level informal overview of the proof, in Section A.1. Then, in Section A.2, we make use of two functions namely, the threshold $\theta(x) = I_{x\geq 0}$ and positive part $x_+ = \max(0,x)$ functions. These two functions are part of the closure of the set ${}_{c,n}\hat{\mathcal{N}}_{++}$ since

$$\begin{aligned} \theta(x) &= \lim_{t\to\infty} h(tx), \\ x_+ &= \lim_{t\to\infty} \zeta(tx). \end{aligned}$$

This extended class of functions that includes $\theta(x)$ and x_+ shall be referred to as ${}_{c,n}\hat{\mathcal{N}}^{\infty}_{++}$. In Section A.3, we give an illustration of the constructive algorithm used to prove universal approximation. Now the proof, as it is stated in Section A.2, only involves functions $\theta(x)$ and x_+, that is, the limit cases of the class ${}_{c,n}\hat{\mathcal{N}}^{\infty}_{++}$ which are actually not part of class ${}_{c,n}\hat{\mathcal{N}}_{++}$. Functions $\theta(x)$ and x_+ assume the use of parameters of infinite value, making the proof without any practical bearing. For this reason, in Section A.4, we broaden the theorem's application from ${}_{c,n}\hat{\mathcal{N}}^{\infty}_{++}$ to ${}_{c,n}\hat{\mathcal{N}}_{++}$, building upon the proof of Section A.2.

A.1 Outline of the Proof

The proof of the first main part (Section A.2) works by construction: we start by setting the approximating function equal to a constant function. Then, we build a grid over the domain of interest and scan through it. At every point of the grid we add a term to the approximating function. This term is a function itself that has zero value at every point of the grid that has already been visited. Thus, this term only affects the current point being visited and some of the points to be visited. The task is therefore to make sure the term being added is such that the approximating function matches the actual function at the point being visited. The functions to be added are chosen from the set ${}_{c,n}\hat{\mathcal{N}}^{\infty}_{++}$ so that each of them individually respects the constraints on the derivatives. The bulk of the work in the proof is to show that, throughout the process, at each scanned point, we need to add a positive term to match the approximating function to the true function. For illustrative purposes, we consider the particular case of call options of Section A.3.

In the second part (Section A.4), we build upon the proof of the first part. The same constructive algorithm is used with the same increment values. We simply consider sigmoidal and softplus functions that are greater or equal, in every point, than their limit counterparts, used in the first part. Products of these softplus and sigmoidal functions are within ${}_{c,n}\hat{\mathcal{N}}_{++}$. Consequently, the function built here is always greater than or equal to its counterpart of the first main part. The main element of the second part is that the difference between these two functions, *at gridpoints*, is capped. This is done by setting the sigmoid and softplus parameter values appropriately. Universality of approximation follows from (1) the capped difference, at gridpoints, between the functions obtained in the first and second parts, (2) the exact approximation obtained at gridpoints in the first part and (3) the bounded function variation between gridpoints.

A.2 Proof of the Universality Theorem for Class ${}_{c,n}\hat{\mathcal{N}}^{\infty}_{++}$

Let D be the compact domain over which we wish to obtain an approximation error below ε in every point. Suppose the existence of an oracle allowing us to evaluate the function in a certain number of points. Let T be the smallest hyperrectangle encompassing D. Let us partition T in hypercubes with sides of length L so that the variation of the function between two arbitrary points of any hypercube is bounded by $\varepsilon/2$. For example, given s, an upper bound on the derivative of the function in any direction, setting $L \leq \frac{\varepsilon}{2s\sqrt{n}}$ would do the trick. Since we have assumed the function to be approximated is Lipschitz, then its derivative is bounded and s does exist. The number of gridpoints is N_1+1 over the x_1 axis, N_2+1 over the x_2 axis, ..., N_n+1 over the x_n axis. Thus, the number of points on the grid formed within T is $H=(N_1+1)\cdot(N_2+1)\cdot\ldots\cdot(N_n+1)$. We define gridpoints $\vec{a}=(a_1,a_2,\cdots,a_n)$ and $\vec{b}=(b_1,b_2,\cdots,b_n)$ as the innermost (closest to origin) and outermost corners of T, respectively. Figure 2 illustrates these values. The points of the grid are defined as:

$$\begin{aligned}
\vec{p}_1 &= a, \\
\vec{p}_2 &= (a_1, a_2, \ldots, a_n + L), \\
\vec{p}_{N_n+1} &= (a_1, a_2, \ldots, b_n), \\
\vec{p}_{N_n+2} &= (a_1, a_2, \ldots a_{n-1} + L, a_n), \\
\vec{p}_{N_n+3} &= (a_1, a_2, \ldots a_{n-1} + L, a_n + L), \ldots, \\
\vec{p}_{(N_n+1)(N_{n-1}+1)} &= (a_1, a_2, \ldots, a_{n-2}, b_{n-1}, b_n), \\
\vec{p}_{(N_n+1)(N_{n-1}+1)+1} &= (a_1, a_2, \ldots, a_{n-2} + L, a_{n-1}, a_n), \ldots, \\
\vec{p}_H &= b.
\end{aligned} \tag{14}$$

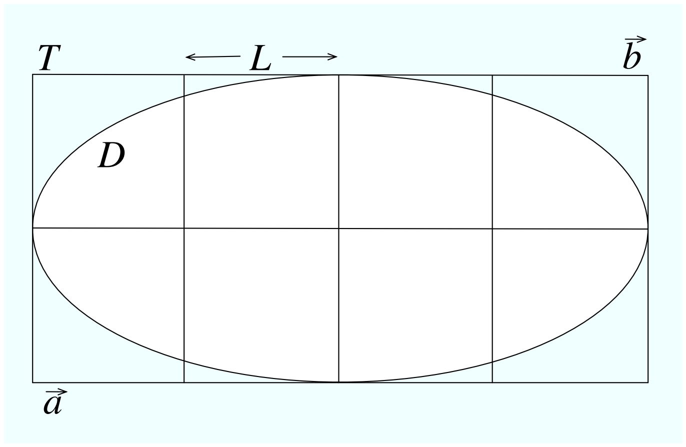

Figure 2: Two dimensional illustration of the proof of universality: ellipse D corresponds to the domain of observation over which we wish to obtain a universal approximator. Rectangle T encompasses D and is partitioned in squares of length L. Points $\vec{a}$ and $\vec{b}$ are the innermost (closest to origin) and outermost corners of T, respectively.

We start with an approximating function $\hat{f}_0 = f(\vec{a})$, that is, the function $\hat{f}_0$ is initially set to a constant value equal to $f(\vec{a})$ over the entire domain. Note that, for the remainder of the proof, notations $\hat{f}_h$, f_h, $\hat{g}_h$, without any argument, refer to the functions themselves. When an argument is present, such as in $f_h(\vec{p})$, we refer to the value of the function f_h evaluated at point $\vec{p}$.

After setting $\hat{f}_0$ to its initial value, we scan the grid according to the order defined in Equation (14). At each point along the grid, we add a term ($\hat{g}_h$, a function) to the current approximating function so that it becomes exact at point $\{\vec{p}_h\}$:

$$\begin{aligned}
\hat{f}_h &= \hat{g}_h + \hat{f}_{h-1}, \\
&= \sum_{k=0}^{h} \hat{g}_k,
\end{aligned}$$

where we have set $\hat{g}_0 = \hat{f}_0$.

The functions $\hat{f}_h$, $\hat{g}_h$ and $\hat{f}_{h-1}$ are defined over the whole domain and the increment function $\hat{g}_h$ must be such that at point $\vec{p}_h$, we have $\hat{f}_h(\vec{p}_h) = f(\vec{p}_h)$. We compute the constant term δ_h as the difference between the value of the function evaluated at point $\vec{p}_h$, $f(\vec{p}_h)$, and the value of the currently accumulated approximating function at the same point $\hat{f}_{h-1}(\vec{p}_h)$:

$$\delta_h = f(\vec{p}_h) - \hat{f}_{h-1}(\vec{p}_h).$$

Now, the function $\hat{g}_h$ must not affect the value of the approximating function at gridpoints that have already been visited. According to our sequencing of the gridpoints, this corresponds to having $\hat{g}_h(\vec{p}_k) = 0$ for $0 < k < h$. Enforcing this constraint ensures that $\forall k \leq h$, $\hat{f}_h(\vec{p}_k) = \hat{f}_k(\vec{p}_k) = f(\vec{p}_k)$. We define

$$\hat{\beta}_h(\vec{p}_k) = \prod_{j=1}^{c} (p_k(j) - p_h(j) + L)_+ / L \cdot \prod_{j=c+1}^{n} \theta(p_k(j) - p_h(j)), \tag{15}$$

where $p_k(j)$ is the j^{th} coordinate of $\vec{p}_k$ and similarly for $\vec{p}_h$. We have assumed, without loss of generality, that the convex dimensions are the first c ones. One can readily verify that $\hat{\beta}_h(\vec{p}_k) = 0$ for $0 < k < h$ and $\hat{\beta}_h(\vec{p}_h) = 1$. We can now define the incremental function as:

$$\hat{g}_h(\vec{p}) = \delta_h \hat{\beta}_h(\vec{p}), \tag{16}$$

so that after all gridpoints have been visited, our final approximation is

$$\hat{f}_H(\vec{p}) = \sum_{h=0}^{H} \hat{g}_h(\vec{p}),$$

with $f(\vec{p}) = \hat{f}_H(\vec{p})$ for all gridpoints.

So far, we have devised a way to approximate the target function as a sum of terms from the set ${}_{c,n}\hat{\mathcal{N}}^{\infty}_{++}$. We know our approximation to be exact in every point of a grid and that the grid is tight enough so that the approximation error is bounded above by $\varepsilon/2$ anywhere within T (thus within D): take any point $\vec{q}$ within a hypercube. Let $\vec{q}_1$ and $\vec{q}_2$ be the innermost (closest to origin) and outermost gridpoints of $\vec{q}$'s hypercube, respectively. Then, we have $f(\vec{q}_1) \leq f(\vec{q}) \leq f(\vec{q}_2)$ and, *assuming* $\delta_h \geq 0\ \forall h$, $f(\vec{q}_1) = \hat{f}_H(\vec{q}_1) \leq \hat{f}_H(\vec{q}) \leq \hat{f}_H(\vec{q}_2) = f(\vec{q}_2)$. Thus, $|\hat{f}_H(\vec{q}) - f(\vec{q})| \leq |f(\vec{q}_2) - f(\vec{q}_1)| \leq Ls\sqrt{n} \leq \varepsilon/2$, since we have set $L \leq \frac{\varepsilon}{2s\sqrt{n}}$. And there remains to be shown that, effectively, $\delta_h \geq 0\ \forall h$. In order to do so, we will express the target function at gridpoint $\vec{p}_h$, $f(\vec{p}_h)$ in terms of the δ_k coefficients ($0 < k \leq h$), then solve for δ_h and show that it is necessarily positive.

First, let $p_k(j) = a(j) + \vec{\imath}_k(j)L$ and define $\vec{\imath}_k = (i_k(1), i_k(2), \ldots, i_k(n))$ so that $\vec{p}_k = \vec{a} + L \cdot \vec{\imath}_k$. Now, looking at Equations (15) and (16), we see that $\hat{g}_k(\vec{p})$ is equal to zero if, for any j, $p_k(j) > p(j)$. Conversely, $\hat{g}_k(\vec{p})$ can only be different from zero if $p_k(j) \leq p(j), \forall j$ or, equivalently, if $i_k(j) \leq i(j), \forall j$.

Next, in order to facilitate the derivations to come, it will be convenient to define some subsets of $\{1, 2, \ldots, H\}$, the indices of the gridpoints of T. Given index h, define $\mathcal{Q}_{h,l} \subset \{1, 2, \ldots, H\}$ as

$$\mathcal{Q}_{h,l} = \{k : i_k(j) \leq i_h(j) \text{ if } j \leq l \text{ and } i_k(j) = i_h(j) \text{ if } j > l\}.$$

In particular, $Q_{h,n} = \{k : i_k(j) \leq i_h(j) \forall j\}$ and $Q_{h,0} = \{h\}$. Thus, we have

$$\begin{aligned}
f(\vec{p}_h) &= \hat{f}_H(\vec{p}_h) \\
&= \sum_{k \in Q_{h,n}} \hat{g}_k(\vec{p}_h) \\
&= \sum_{k \in Q_{h,n}} \delta_k \prod_{j=1}^{c} (i_h(j) - i_k(j) + 1)_+ \prod_{j=c+1}^{n} \theta(i_h(j) - i_k(j)) \\
&= \sum_{k \in Q_{h,n}} \delta_k \prod_{j=1}^{c} (i_h(j) - i_k(j) + 1). \qquad (17)
\end{aligned}$$

Now, let us define the finite difference of the function along the l^{th} axis as

$$\Delta_l f(\vec{p}_h) = f(\vec{p}_h) - f(\vec{p}_{h_l}), \qquad (18)$$

where $\vec{p}_{h_l}$ is the neighbor of $\vec{p}_h$ on T with all coordinates equal except along the l^{th} axis where $i_{h_l}(l) = i_h(l) - 1$. The following relationship shall be useful:

$$\begin{aligned}
Q_{h,l} \setminus Q_{h_l,l} &= \{k : i_k(j) \leq i_h(j) \text{ if } j < l, i_k(l) \leq i_h(l) \text{ and } i_k(j) = i_h(j) \text{ if } j > l\} \setminus \\
&\quad \{k : i_k(j) \leq i_h(j) \text{ if } j < l, i_k(l) < i_h(l) \text{ and } i_k(j) = i_h(j) \text{ if } j > l\} \\
&= \{k : i_k(j) \leq i_h(j) \text{ if } j \leq l-1 \text{ and } i_k(j) = i_h(j) \text{ if } j > l-1\} \\
&= Q_{h,l-1}. \qquad (19)
\end{aligned}$$

We now have the necessary tools to solve for δ_h by differentiating the target function. Using Equations (17), (18) and (19) we get:

$$\begin{aligned}
\Delta_n f(\vec{p}_h) &= \sum_{k \in Q_{h,n}} \delta_k \prod_{j=1}^{c} (i_h(j) - i_k(j) + 1) \\
&\quad - \sum_{k \in Q_{h_n,n}} \delta_k \prod_{j=1}^{c} (i_{h_n}(j) - i_k(j) + 1).
\end{aligned}$$

Since $i_{h_n}(j) = i_h(j)$ for $j \leq c$, then

$$\Delta_n f(\vec{p}_h) = \sum_{k \in Q_{h,n-1}} \delta_k \prod_{j=1}^{c} (i_h(j) - i_k(j) + 1).$$

This process is repeated for non-convex dimensions $n-1, n-2, \ldots, c+1$ until we obtain

$$\Delta_{c+1} \ldots \Delta_n f(\vec{p}_h) = \sum_{k \in Q_{h,c}} \delta_k \prod_{j=1}^{c} (i_h(j) - i_k(j) + 1),$$

at which point we must consider differentiating with respect to convex dimensions:

$$\begin{aligned}
\Delta_c \ldots \Delta_n f(\vec{p}_h) &= \sum_{k \in Q_{h,c}} \delta_k \prod_{j=1}^{c} (i_h(j) - i_k(j) + 1) \\
&\quad - \sum_{k \in Q_{h_c,c}} \delta_k \prod_{j=1}^{c} (i_{h_c}(j) - i_k(j) + 1) \\
&= \sum_{k \in Q_{h,c}} \delta_k (i_h(c) - i_k(c) + 1) \prod_{j=1}^{c-1} (i_h(j) - i_k(j) + 1) \\
&\quad - \sum_{k \in Q_{h_c,c}} \delta_k (i_h(c) - i_k(c)) \prod_{j=1}^{c-1} (i_h(j) - i_k(j) + 1).
\end{aligned}$$

According to Equation (19), $Q_{h,c} \backslash Q_{h_c,c} = Q_{h,c-1}$ and by definition $i_k(c) - i_h(c) = 0 \; \forall k \in Q_{h,c-1}$. Using this, we subtract a sum of zero terms from the last equation in order to simplify the result:

$$\begin{aligned}
\Delta_c \ldots \Delta_n f(\vec{p}_h) &= \sum_{k \in Q_{h,c}} \delta_k (i_h(c) - i_k(c) + 1) \prod_{j=1}^{c-1} (i_h(j) - i_k(j) + 1) \\
&\quad - \sum_{k \in Q_{h_c,c}} \delta_k (i_h(c) - i_k(c)) \prod_{j=1}^{c-1} (i_h(j) - i_k(j) + 1) \\
&\quad - \sum_{k \in Q_{h,c-1}} \delta_k (i_h(c) - i_k(c)) \prod_{j=1}^{c-1} (i_h(j) - i_k(j) + 1) \\
&= \sum_{k \in Q_{h,c}} \delta_k (i_h(c) - i_k(c) + 1) \prod_{j=1}^{c-1} (i_h(j) - i_k(j) + 1) \\
&\quad - \sum_{k \in Q_{h,c}} \delta_k (i_h(c) - i_k(c)) \prod_{j=1}^{c-1} (i_h(j) - i_k(j) + 1) \\
&= \sum_{k \in Q_{h,c}} \delta_k \prod_{j=1}^{c-1} (i_h(j) - i_k(j) + 1).
\end{aligned}$$

Differentiating once again with respect to dimension c:

$$\begin{aligned}
\Delta_c^2 \ldots \Delta_n f(\vec{p}_h) &= \sum_{k \in Q_{h,c}} \delta_k \prod_{j=1}^{c-1} (i_h(j) - i_k(j) + 1) \\
&\quad - \sum_{k \in Q_{h_c,c}} \delta_k \prod_{j=1}^{c-1} (i_{h_c}(j) - i_k(j) + 1).
\end{aligned}$$

and since $i_{h_c}(j) = i_h(c) \forall j \leq c-1$, then

$$\Delta_c^2 \ldots \Delta_n f(\vec{p}_h) = \sum_{k \in Q_{h,c-1}} \delta_k \prod_{j=1}^{c-1} (i_h(j) - i_k(j) + 1).$$

This process of differentiating twice is repeated for all convex dimensions so that

$$\begin{aligned}\Delta_1^2 \ldots \Delta_c^2 \Delta_{c+1} \ldots \Delta_n f(\vec{p}_h) &= \sum_{k \in Q_{h,0}} \delta_k. \\ &= \delta_h\end{aligned}$$

Now, by definition of the integral operator,

$$\Delta f = \frac{f(b)-f(a)}{b-a} = \frac{1}{b-a}\int_a^b f'\,dx,$$

so that if $f' \geq 0$ over the range $[a,b]$, then consequently, $\Delta f \geq 0$. Since, according to Equation (6), we have

$$\frac{\partial^{n+c} f(p_h)}{\partial x_1^2 \partial x_2^2 \ldots \partial x_c^2 \partial x_{c+1} \ldots \partial x_n} \geq 0,$$

then $\Delta_1^2 \ldots \Delta_c^2 \Delta_{c+1} \ldots \Delta_n f(\vec{p}_h) \geq 0$ and $\delta_h \geq 0$.

For gridpoints with either $i_h(j) = 1$ for any j or with $i_h(j) = 2$ for any $j \leq c$, solving for δ_h requires fewer than $n+c$ differentiations. Since the positivity of the derivatives of f corresponding to these lower order differentiations are covered by Equation (6), then we also have that $\delta_h \geq 0$ for these gridpoints laying at or near some of the boundaries of T. Thus, ${}_{c,n}\hat{\mathcal{N}}_{++}^{\infty}$ is a universal approximator of ${}_{c,n}\hat{\mathcal{F}}_{++}$.

$\square$

A.3 Illustration of the Constructive Algorithm

In order to give the reader a better intuition regarding the constructive algorithm and as how to solve for δ_h, we apply the developments of the previous subsection to ${}_{1,2}\hat{\mathcal{N}}_{++}$, the set of functions that include call price functions, that is, positive convex w.r.t. the first variable and monotone increasing w.r.t. both variables. Figure 3 illustrates the two dimensional setting of our example with the points of the grid labelled in the order in which they are scanned according the constructive procedure. Here, we will solve δ_6.

For the set ${}_{1,2}\hat{\mathcal{N}}_{++}$, we have,

$$f(\vec{p}_h) = \sum_{k=1}^{H} \delta_k \cdot (p_h(1) - p_k(1) + L)_+ \cdot \theta(p_h(2) - p_k(2)).$$

Applying this to the six gridpoints of Figure 3, we obtain $f(\vec{p}_1) = \delta_1$, $f(\vec{p}_2) = (\delta_1 + \delta_2)$, $f(\vec{p}_3) = (2\delta_1 + \delta_3)$, $f(\vec{p}_4) = (2\delta_1 + 2\delta_2 + \delta_3 + \delta_4)$, $f(\vec{p}_5) = (3\delta_1 + 2\delta_3 + \delta_5)$, $f(\vec{p}_6) = (3\delta_1 + 3\delta_2 + 2\delta_3 + 2\delta_4 + \delta_5 + \delta_6)$.

Differentiating w.r.t. the second variable, then the first, we have:

$$\begin{aligned}\Delta_2 f(\vec{p}_6) &= f(\vec{p}_6) - f(\vec{p}_5) \\ \Delta_1\Delta_2 f(\vec{p}_6) &= (f(\vec{p}_6) - f(\vec{p}_5)) - (f(\vec{p}_4) - f(\vec{p}_3)) \\ \Delta_1^2\Delta_2 f(\vec{p}_6) &= (f(\vec{p}_6) - f(\vec{p}_5)) - (f(\vec{p}_4) - f(\vec{p}_3)) \\ &\quad - (f(\vec{p}_4) - f(\vec{p}_3)) + (f(\vec{p}_2) - f(\vec{p}_1)) \\ &= \delta_6.\end{aligned}$$

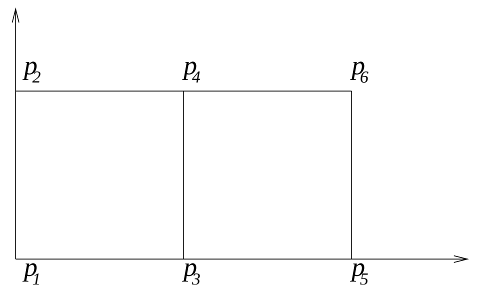

Figure 3: Illustration in two dimensions of the constructive proof. The points are labelled according to the order in which they are visited. The function is known to be convex w.r.t. to the first variable (abscissa) and monotone increasing w.r.t. both variables.

The conclusion associated with this result is that the third finite difference of the function must be positive in order for δ_6 to be positive as well. As stated above, enforcing the corresponding derivative to be positive is a stronger condition which is respected by all element functions of ${}_{c,n}\hat{\mathcal{N}}_{++}$. In the illustration above, other increment terms (δ_1 through δ_5) can be solved for with fewer differentiations. As mention in the previous subsection, derivatives associated to these lower order differentiations are all positive.

A.4 Proof of the Universality Theorem for Class ${}_{c,n}\hat{\mathcal{N}}_{++}$

In Section A.2, we obtained an approximating function $\hat{f}_H \in {}_{c,n}\hat{\mathcal{N}}^{\infty}_{++}$ such that $|\hat{f}_H - f| \leq \varepsilon/2$. Here, we will build a function $\tilde{f}_H \in {}_{c,n}\hat{\mathcal{N}}_{++}$ everywhere greater or equal to $\hat{f}_H$, but we will show how the difference between the two functions can be bounded so that $\tilde{f}_H - \hat{f}_H \leq \varepsilon/2$ at all gridpoints.

We start with an approximating function $\tilde{f}_0 = \hat{f}_0 = f(\vec{a})$, that is, $\tilde{f}_0$ is initially set to a constant value equal to $f(\vec{a})$ over the entire domain. Then, we scan the grid in an orderly manner, according to the definition of the set of points $\{\vec{p}_h\}$. At each point $\vec{p}_h$ along the grid, we add a term $\tilde{g}_h$ (a function) to the current approximating function $\tilde{f}_{h-1}$:

$$\begin{aligned}\tilde{f}_h &= \tilde{g}_h + \tilde{f}_{h-1}\\ &= \sum_{k=1}^{h} \tilde{g}_k\\ &= \sum_{k=1}^{h} \delta_k \tilde{\beta}_k,\end{aligned}$$

where the δ_k are kept equal to the ones found in Section A.2 and we define the set of $\tilde{\beta}_k$ functions as a product of sigmoid and softplus functions, one for each input dimension:

$$\tilde{\beta}_h(\vec{p}) = \prod_{j=1}^{n} \tilde{\beta}_{h,j}(\vec{p}).$$

For each of the convex coordinates, we set:

$$\tilde{\beta}_{h,j}(\vec{p}) = \frac{1}{\alpha}\zeta(\alpha\cdot(p(j) - p_h(j) + L)). \tag{20}$$

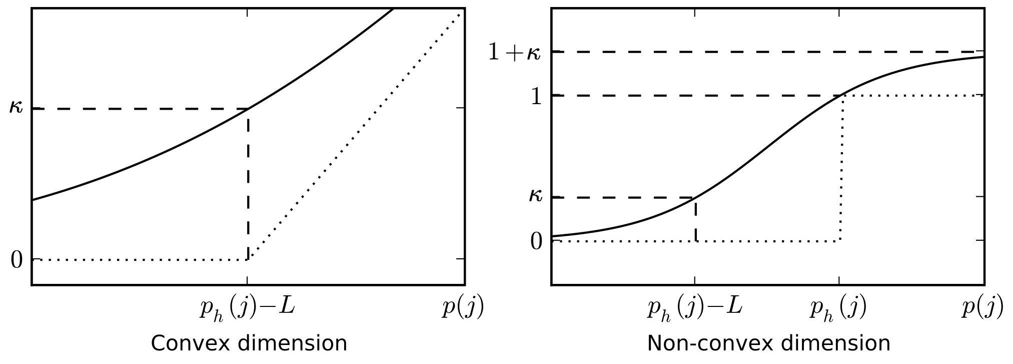

Figure 4: Illustration of the difference between $\tilde{\beta}_{h,j}$ (solid) and $\hat{\beta}_{h,j}$ (dotted) for convex (left) and non-convex (right) dimensions.

where $\alpha > 0$. Now, note that κ, the maximum of the difference between the softplus function of Equation (20) and the positive part function $\hat{\beta}_{h,j}(\vec{p}) = (p(j) - p_h(j) + L)_+$, is attained for $p(j) = p_h(j) - L$ where the difference is $\ln 2/\alpha$. Thus, in order to cap the difference resulting from the approximation along the convex dimensions, we simply need to set κ (α) to a small (large) enough value which we shall soon define. Let us now turn to the non-convex dimensions where we set:

$$\tilde{\beta}_{h,j}(\vec{p}) \quad = \quad (1+\kappa)\, h(\gamma \cdot p(j) + \eta)$$

and add two constraints:

$$\begin{aligned} h(\gamma(p_h(j) - L) + \eta) &= \frac{\kappa}{1+\kappa}, \\ h(\gamma \cdot p_h(j) + \eta) &= \frac{1}{1+\kappa}. \end{aligned}$$

Solving for γ and η, we obtain:

$$\gamma \quad = \quad -\frac{2}{L} \ln \kappa, \tag{21}$$

$$\eta \quad = \quad \left(\frac{2p_h(j)}{L} - 1 \right) \ln \kappa. \tag{22}$$

For non-convex dimensions, we have $\hat{\beta}_{h,j}(\vec{p}) = \theta(p(j) - p_h(j))$. Thus, for values of $p(j)$ such that $p_h(j) - L < p(j) < p_h(j)$, we have a maximum difference $\tilde{\beta}_{h,j} - \hat{\beta}_{h,j}$ of 1. For other values of $p(j)$, the difference is capped by κ. In particular, the difference is bounded above by κ for all gridpoints and is zero for gridpoints with $p(j) = p_h(j)$. These values are illustrated in Figure 4.

We now compare incremental terms. Our goal is to cap the difference between $\tilde{g}_h$ and $\hat{g}_h$ by $\varepsilon/2H$. This will lead us to bound the value of κ. At gridpoints, $\hat{\beta}_h$ is equal to

$$\hat{\beta}_h \quad = \quad \prod_{j=1}^{n} m_j,$$

where $m_j \in \{0,1\}$ along non-convex dimensions and m_j is equal to a non-negative integer along the convex dimensions. Also, since $\tilde{\beta}_{h,j} - \hat{\beta}_{h,j} \leq \kappa$ at gridpoints, then

$$\tilde{\beta}_h \leq \prod_{j=1}^{n}(m_j+\kappa).$$

In order find a bound on the value of κ, we need to consider two cases. First, consider the case where $m_j > 0 \, \forall j$:

$$\begin{aligned}
\tilde{g}_h - \hat{g}_h &\leq \delta_h \left(\prod_{j=1}^{n}(m_j+\kappa) - \prod_{j=1}^{n} m_j \right) \\
&\leq \delta_h \left(\prod_{j=1}^{n} m_j(1+\kappa) - \prod_{j=1}^{n} m_j \right) \\
&= \delta_h((1+\kappa)^n - 1) \prod_{j=1}^{n} m_j \\
&\leq \delta_h((1+\kappa)^n - 1) \prod_{j=1}^{n} (N_j+1) \\
&\leq \varepsilon((1+\kappa)^n - 1)H.
\end{aligned}$$

In that case, we have $\tilde{g}_h - \hat{g}_h < \varepsilon/2H$ if

$$\kappa \leq (1/2H^2+1)^{1/n} - 1. \tag{23}$$

Now, consider cases where $\exists j : m_j = 0$. Let $d = \#\{j : m_j = 0\}$. Then,

$$\begin{aligned}
\tilde{g}_h - \hat{g}_h &\leq \delta_h \left(\prod_{j=1}^{n}(m_j+\kappa) - \prod_{j=1}^{n} m_j \right) \\
&\leq \delta_h \kappa^d \prod_{m_j \neq 0} (N_j+1)(1+\kappa) \\
&\leq \delta_h \kappa^d (1+\kappa)^{n-d} H \\
&\leq \varepsilon \kappa^d 2^{n-d} H \\
&\leq \varepsilon \kappa 2^{n-1} H,
\end{aligned}$$

so that here, the bound on κ is:

$$\kappa \leq \frac{1}{2^n H^2}. \tag{24}$$

Depending on the relative values of n and H, one of the two bounds may be effective so that both values of Equations (23) and (24) must be considered in order to set an upper bound on κ:

$$\kappa \leq \min\left((1/2H^2+1)^{1/n} - 1, \frac{1}{2^n H^2} \right).$$

Values for $\alpha = \ln 2/\kappa$, γ, and η (Equations 21 and 22) are derived accordingly.

Thus, for any gridpoint, we have:

$$\begin{aligned}\tilde{f}_h - \hat{f}_h &= \sum_{k=1}^{H} \tilde{g}_h - \hat{g}_h \\ &\leq H \cdot \varepsilon/2H \\ &= \varepsilon/2.\end{aligned}$$

In Section A.2, we developed an algorithm such that $\hat{f} = f$ for any gridpoint. In this present subsection, we showed that softplus and sigmoid parameters could be chosen such that $\hat{f} \leq \tilde{f} \leq \hat{f} + \varepsilon/2$ for any gridpoint. Note that $f, \hat{f}$, and $\tilde{f}$ are increasing along each input dimension.

As in Section A.2, consider any point $\vec{q} \in D$. Let $\vec{q}_1$ and $\vec{q}_2$ be the innermost and outermost gridpoints of $\vec{q}$'s encompassing hypercube of side length L. In Section A.2, we showed how a grid could be made tight enough so that $f(\vec{q}_2) - f(\vec{q}_1) \leq \varepsilon/2$.

With these results at hand, we can set upper and lower bounds on $\tilde{f}(\vec{q})$. First, observe that $\tilde{f}_H(\vec{q}) \geq \tilde{f}_H(\vec{q}_1) \geq \hat{f}_H(\vec{q}_1) = f(\vec{q}_1)$, which provides us with a lower bound on $\tilde{f}_H(\vec{q})$. Next, for the upper bound we have: $\tilde{f}_H(\vec{q}) \leq \tilde{f}_H(\vec{q}_2) \leq \hat{f}_H(\vec{q}_2) + \varepsilon/2 = f(\vec{q}_2) + \varepsilon/2 \leq f(\vec{q}_1) + \varepsilon$. Thus, $\tilde{f}_H(\vec{q}) \in [f(\vec{q}_1), f(\vec{q}_1) + \varepsilon]$ and $f(\vec{q}) \in [f(\vec{q}_1), f(\vec{q}_1) + \varepsilon/2] \subset [f(\vec{q}_1), f(\vec{q}_1) + \varepsilon]$. Since both $f(\vec{q})$ and $\tilde{f}(\vec{q})$ are within a range of length ε, then $|\tilde{f}(\vec{q}) - f(\vec{q})| \leq \varepsilon$.

$\square$

References

A. R. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, 39(3):930–945, 1993.

F. Black and M. Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 81(3):637–654, 1973.

G. Cybenko. Continuous valued neural networks with two hidden layers are sufficient. Technical report, Department of Computer Science, Tufts University, Medford, MA, 1988.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2:303–314, 1989.

M.C. Delfour and J.-P. Zolésio. *Shapes and Geometries: Analysis, Differential Calculus, and Optimization*. SIAM, 2001.

C. Dugas, O. Bardou, and Y. Bengio. Analyses empiriques sur des transactions d'options. Technical Report 1176, Départment d'informatique et de Recherche Opérationnelle, Université de Montréal, Montréal, Québec, Canada, 2000.

R. Garcia and R. Gençay. Pricing and hedging derivative securities with neural networks and a homogeneity hint. Technical Report 98s-35, CIRANO, Montréal, Québec, Canada, 1998.

K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

J.M. Hutchinson, A.W. Lo, and T. Poggio. A nonparametric approach to pricing and hedging derivative securities via learning networks. *Journal of Finance*, 49(3):851–889, 1994.

M. Leshno, V. Lin, A. Pinkus, and S. Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6:861–867, 1993.

J. Moody. *Prediction Risk and Architecture Selection for Neural Networks*. Springer, 1994.